A Balanced Encoding Technique to Reduce Stray Current Impacts in Crossbar Memories

Adam C. Cabe, *Student Member, IEEE,* Garrett S. Rose, *Member, IEEE,*
Mircea R. Stan, *Senior Member, IEEE*

Abstract

Nanoscale crossbar array memory technologies exhibit many promising features including ultra-dense bit-cell arrays, regularity, and ease of fabrication. However, exploiting these features has proven difficult due to stray current paths in the crossbar array topology. The stray currents arise from off-path currents through non-ideal devices used within the memory arrays (Magnetic-Tunnel-Junctions, Phase-Change devices, molecules). This work presents a solution to reduce the impact of such stray currents, combining both a circuit and balanced data-encoding technique. Using this technique, memories can scale out beyond array sizes of 64x64, allowing the creation of larger total memory structures. This work discusses the overhead of the proposed circuit/encoding scheme, and compares the final solution to the common 1T-1MTJ Magnetic Random-Access-Memory (MRAM) design.

Index Terms

Crossbar Array, Nanoelectronics, Resistive Memory, MRAM, Encoding.

I. Introduction

The push to scale IC design to and beyond the limits of CMOS integration presents many new, exciting, and challenging areas of VLSI research. The International Technology Roadmap for Semiconductors (ITRS) states two primary challenges for scaling IC technologies to and beyond the end of the CMOS roadmap. The first is to extend ”CMOS beyond its ultimately scaled density and functionality by integrating, for example, a new high speed, dense, and low power memory technology on the CMOS platform. [1]” The second challenge is to ”extend information processing substantially beyond that attainable by CMOS alone using an innovative combination of new devices and architectural approaches for extending CMOS, and eventually, inventing a new information processing platform. [1]”

These challenges stress the need to develop new memory and logic architectures, that are not only low-power, but are reliable and have the capability of outperforming the current, state-of-art CMOS based architectures. This paper

A. Cabe and M. Stan are with the Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA, 22903, G. Rose is with the Department of Electrical and Computer Engineering, Brooklyn Poly, NY. Primary contact author e-mail: acc9x@virginia.edu.
This work was funded through a grant from the MARCO Interconnect Focus Center. Portions of this work were originally presented at the 2007 IEEE Conference on Nanotechnology.

Manuscript received February 11, 2008

focuses on emerging memory technologies, such as MRAM, phase change memory (PCM), molecular memory, and metal-oxide memory. Each of the examined memory technologies is non-volatile, which is an intrinsic benefit over their CMOS memory counterparts SRAM and DRAM. Furthermore, this work focuses on the implementation of such memories in a crossbar array architecture, not the conventional 1T-1D (1-Transistor 1-Device) arrangement.

The crossbar array topology is popular for both its ease in fabrication, primarily due to the structures regularity, and its ultra-dense bit-cell structure. Researchers have achieved crossbar array bit densities of $10^{11}cm^{-2}$, creating wire widths of 16 nm and wire pitches of 33 nm with no observed structural defects [2]. Although much progress has been made in fabricating crossbar array memories, recent works highlight the difficulties in reading the state of each bit in large crossbar arrays [2], [3]. Non-ideal devices lead to "stray current paths" in the memory arrays, which obscure the read output voltage. In certain cases, results show it impossible to differentiate between a logic '1' and a logic '0' for arrays as small as $6x6$ devices [3].

One primary reason that these novel memory architectures migrated more towards the 1T-1D topology is to get away from the stray current problem associated with the crossbar array. Imagine building a full scale 256 kB memory from memory banks as small as 6x6. This would require over 7200 small memory banks, and a massive overhead in terms of routing and bank decoding. The primary goal of this work is to develop a method to eliminate the impact of stray currents on crossbar memories, and allow designers to fully utilize the dense bit-cell structure provided through the crossbar topology.

This work combines a circuit technique with a balanced data encoding method that in conjunction work to reduce the impact of stray currents. Simulation results show that by using the presented techniques, array sizes scale out beyond 64x64 bits. The circuit techniques, encoding algorithm, and hardware implementations are all presented in detail, including the overhead involved with employing such a strategy in crossbar array architectures. The encoding techniques and peripheral decoding circuits are implemented in CMOS, and are simulated using the Predictive Transistor Model (PTM) libraries. Furthermore, the proposed techniques are studied using six different types of nanoscale devices, ranging from magnetic tunnel junctions (MTJ) to molecular switches.

The paper begins by introducing the crossbar array in section II, presenting a clear description of the architecture and the problems with stray currents. Section III presents an overview of the proposed circuit and encoding solutions to reduce the impacts of stray currents. Section IV follows by presenting detailed circuit simulation results of the encoding techniques performance. The encoder/decoder hardware implementation details are presented in section V. Section VI presents the hardware synthesis results for the encoder/decoder circuitry, followed by section VII, which provides a comparison study against a conventional 1T-1MTJ MRAM topology. Section VIII concludes the work and discusses possible future work.

II. Architectural Analysis of Crossbar Memory

A. The Crossbar Memory Array

The crossbar array topology consists of two sets of parallel wires crossing perpendicularly [4], where at each intersection there exists a two-state resistive device. The overall circuit is therefore a two-dimensional array of

two-terminal programmable elements. An example of this structure is shown in Fig. 1A. Programming a junction is performed by applying specified voltages on the two nanowires connecting the device. The programming procedure is different for various devices, however the read procedure is similar for most technologies. The state is read by simply applying a voltage across the selected device and sensing the current through its two terminals. Additionally, the read voltage must be small enough not to disturb the state of the nanodevice.

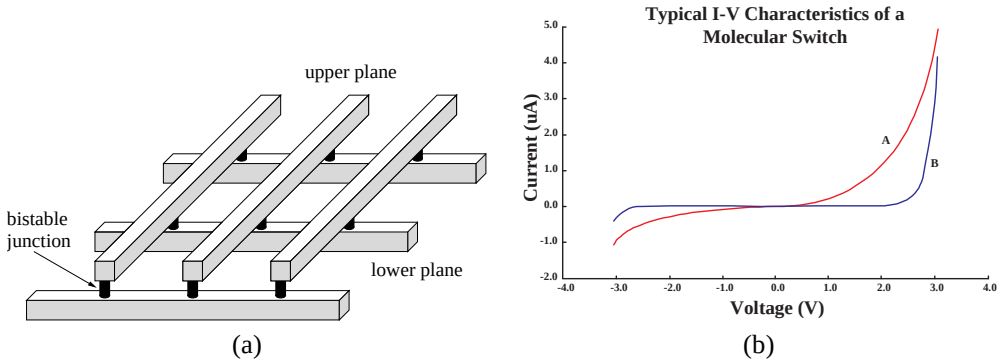


Fig. 1. A) Crosspoint array architecture as used in crosspoint memory. At each wire junction exists a bi-stable resistive switch, making this structure a 2D array of programmable devices [5]. B. Hysteretic I-V characteristics of a typical bi-stable nanodevice. The curves are taken from a Verilog-A model based on experimental data of a molecular device [6].

The I-V characteristic of a typical resistive device is shown adjacent to the crossbar in Fig. 1B. The two separate I-V curves represent the devices high and the low conductivity states, and reflect the bi-stability or hysteresis of the junction. Under the application of a specified toggle voltage, these devices can switch between states - applying a large negative toggle voltage will switch the device into the low conductivity state, while applying a positive toggle voltage will switch the device back into the high conductivity state. The low conductivity state is referred to as ‘off’ since less current flows, while the high conductivity is the ‘on’ state.

The shape of these curves, and the required toggle voltages, depend upon the particular device technology. Several important characteristics embody the efficacy of the nanodevice, including ‘on/off’ current ratio, rectification, size, and required toggle voltages. The device ‘on/off’ ratio is a measure of the current in the on state versus the off state given an applied voltage across the terminals. This ratio can be as small as 2 or larger than 1000 depending on the given device [7]–[14]. Additionally, some devices exhibit intrinsic rectification, which makes their forward bias currents larger than their reverse bias currents. The curves in Fig. 1B show this rectification, and are modeled after an oligo(phenylene ethynylene) (OPE) molecule with a nitro side-group [12].

The advantages of the crossbar array coupled with these nanodevices were briefly discussed in the opening section, and included the ultra-compact bit-cell, the architectures regularity, and the ease of fabrication. However, there still exist intrinsic problems with the crossbar array architecture. The two primary problems within this architecture are how to work around the stray currents within the arrays, and how to integrate the peripheral decoding logic into the finished product. This work assumes the use of CMOS for the peripheral circuitry, and that a technique such as CMOL (Cmos + nanowires + MOLecules) or FPNI (Field-Programmable Nanowire-Interconnect) can be used

to connect the CMOS with the nanowire fabric [15]–[17]. The focus of this work is to eliminate the stray current impacts from crossbar memories. The next section describes the impacts of stray current paths, and provides some insight into solving this problem.

B. Crossbar Array Functionality and Size Limits

Fig. 2 illustrates a schematic representation of a 4x4 crossbar memory circuit. Fig. 2A shows how to write a selected nanodevice, ignoring the peripheral decoding circuitry, and Fig. 2B shows the read procedure. Programming a junction is done by applying specified voltages on the two nanowires connecting to the device. In this schematic figure, V_{WR} is applied to one terminal of the device, and $-V_{WR}$ is applied to the opposite terminal, so that the equivalent voltage across the device is $2 \cdot V_{WR}$. If the toggle voltage, or the voltage at which the device changes state, is $\pm V_{tog}$, care must be taken to ensure that $V_{WR} < V_{tog} < 2 \cdot V_{WR}$ when programming a junction. This ensures no unselected devices are written [3]. The specific programming procedures differ for varying devices, particularly in terms of write pulse widths and timing, however the general theme remains as depicted in Fig. 2A.

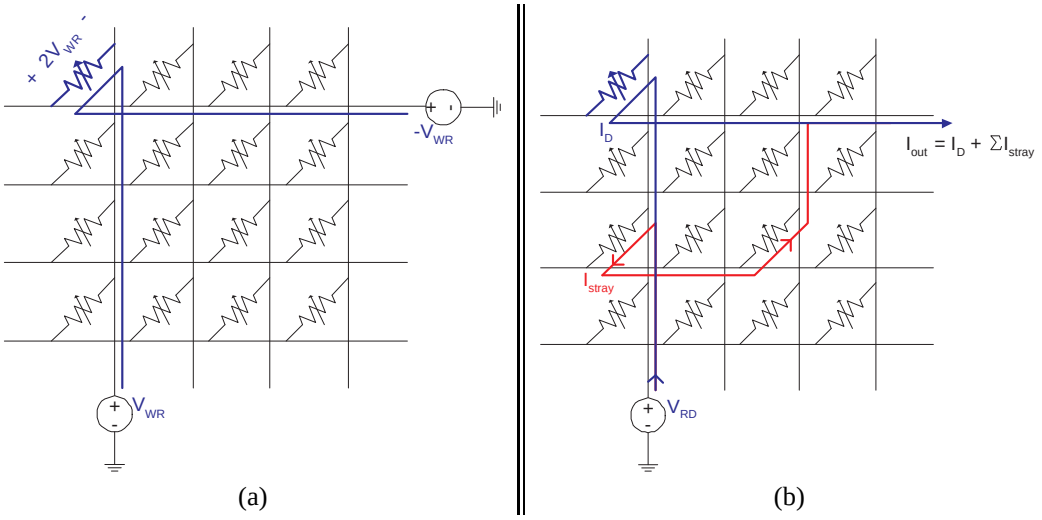


Fig. 2. (a) Schematic of a 4x4 crossbar array. Perform a write function by applying a total of $2 \cdot V_{WR}$ across the nanodevice. (b) Schematic showing the read procedure on the selected device in the upper-left corner. Final output current depends not only on the selected device, but on stray currents in the memory array.

The methods used to read the state of a nano-resistive device are similar among the various technologies. An example read is shown in fig. 2B, where the state is determined by applying a voltage across the device and reading the current through the device. One interesting thing to note when reading the crossbar array, specifically considering when the non-selected rows and columns are electrically floating, the output is not solely dependent upon the selected device. This is illustrated in Fig. 2B, where $I_{out} = I_D + \Sigma I_{stray}$. These stray currents flow through the unselected devices, and make it difficult to determine the device state.

One vital metric for such crossbar array memories is the ratio of the output currents/voltages for logic ‘1’ and logic ‘0’ values, referred to here as the ‘1’/‘0’ current ratio ($F_{1/0} = I_{out1}/I_{out0}$). This is not to be confused with

the ‘on/off’ device current ratio, as that only pertains to single devices. The measure of $F_{1/0}$ looks at the output of the entire memory array. It is desirable to maximize $F_{1/0}$ in order to reliably read a device from the crossbar array, and it is essential that this $F_{1/0}$ ratio be greater than one. In the crossbar array, the presence of stray currents makes it even more vital to have this $F_{1/0} \gg 1$.

Fig. 2B highlights one stray current path, however in actuality stray currents flow through all unselected memory paths. If these nanodevices exhibited 100% rectification then stray currents would not effect the memory, as each off-path device would act like a reverse-bias diode. However, these non-ideal devices do conduct current even when reverse-biased, and these stray current levels depend on both the applied voltage and the data stored in the unselected devices.

Ideally, the output current I_{out} in Fig. 2B should be dependent only on R_D, however, the effect of stray currents, particularly for large arrays, can dominate the output voltage. The stray current levels are data dependent, and depending on the value of the selected bit, may add up in a constructive or destructive way. Given a ‘1’ value on the selected bit, the best case is when all the unselected bits are also ‘1’, and the worst case is when all the unselected bits are ‘0’. For a ‘0’ value on the selected bit, the best case is when all the unselected bits are also ‘0’, and the worst case is when all the unselected bits are ‘1’. Once the worst-case $F_{1/0}$ ratio falls below one, it becomes impossible to differentiate between reading a logic ‘1’ and a ‘0’, and the memory becomes unusable [6].

Previous results demonstrate this effect, where given an array as small as 6x6 it becomes impossible to tell whether a bit stores a logic ‘1’ or logic ‘0’ considering molecular nanoelectronic devices [3], [18]. Furthermore, Green et. al. comment that in their fabricated 160 kb molecular memory, much of the output switching may simply be due to the parasitics and stray currents in the memory changing the output levels [2].

One previous method used to assuage the size limits on crossbar memory is to ground the unselected rows in the crossbar array [6]. This circuit technique increases the overhead of the peripheral circuitry, however does permit the memory to scale near $64x64$ bits, while keeping an $F_{1/0}$ ratio of 2.23. This $F_{1/0}$ is still rather small, as the output voltage difference is only about a factor of 2 between a logic ‘1’ and ‘0’, leaving little margin for error from any device level variations, or mismatch in the sensing circuits.

In summary, determining the state of one bit in a crosspoint memory is challenging since stray currents that flow through unselected devices impact the output. These currents are dependent on the state of the unselected devices, leading to widely different output values for best and worst case logic ‘1’ and logic ‘0’ bits. For crossbar memory technologies to deliver on the promise of providing ultra-dense, reliable architectures, a solution is needed to eliminate the impact of these stray currents. The next sections present such a solution.

III. Eliminating the Impacts of Crossbar Array Stray Currents

A. Circuit Solution

The previous chapter discussed how grounding the unselected rows can help improve the allowable crossbar memory size, as was demonstrated previously [6]. In this section, the simple next step is taken to ground both the unselected rows and columns when reading a particular device. This is schematically shown in Fig. 3A. While

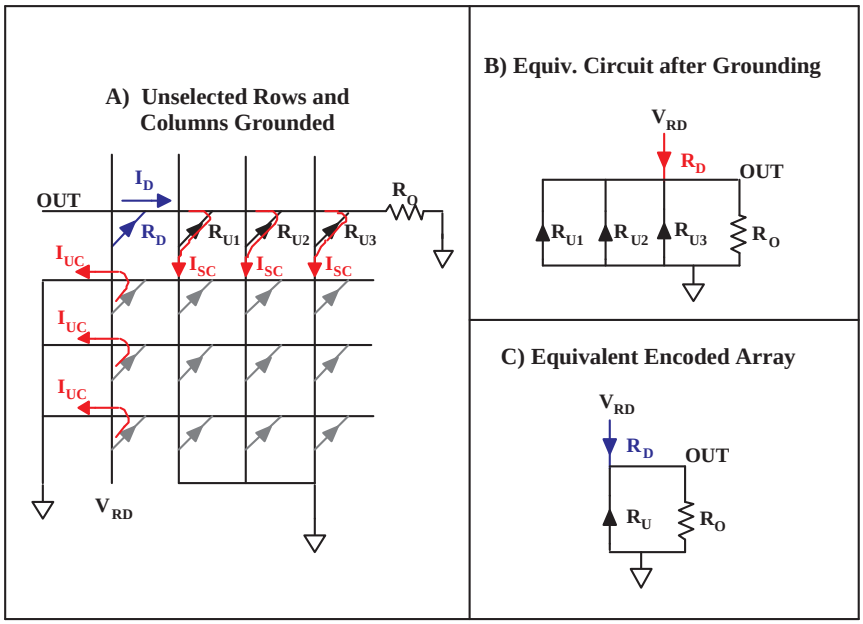


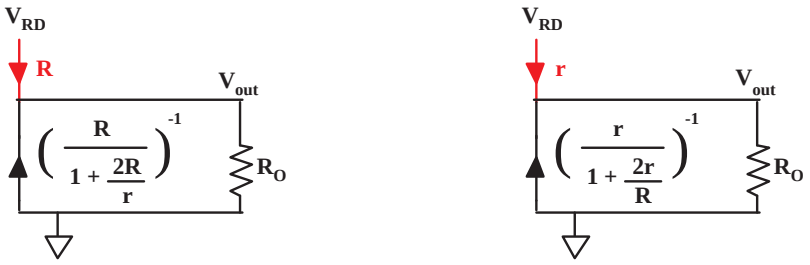
Fig. 3. A. Schematic of crossbar array when grounding unselected rows and columns. Reading selected device R_D. B. Simplified equivalent circuit after grounding unselected rows and columns. C. Desired equivalent circuit, achieved through data encoding.

this is a simple addition to grounding just the unselected rows, it greatly helps reduce the overall stray currents. Furthermore, a load device is added to the output in Fig. 3A, which converts the output into a voltage rather than a current. This helps, as the resistances of such nanodevices can be hundreds of $M\Omega s$, making it difficult to sense these ultra-small current levels. A simplified view of the new circuit while reading the selected device R_D is shown in Fig. 3B.

By grounding both the unselected rows and columns, the output becomes dependent upon only the device being read and the unselected devices on the selected row (R_{U1-3}). As the currents from the unselected rows (I_{UC}'s in Fig. 3A) are shunted to ground before reaching node OUT, they do not impact the output voltage. However, the unselected devices on the selected row directly draw current from node OUT (I_{SC}'s in Fig. 3A), making the output dependent upon their state.

Having the simplified circuit in Fig. 3B means the output voltage is data dependent upon only the selected device R_D and devices R_{U1-3}. Before when grounding only the unselected rows, current could flow out through the devices R_{U1-3} and then through the remaining unselected (grayed) devices to ground. Therefore, the output would still be data dependent on almost every device in the array.

As the output in Fig. 3 is still data dependent upon several unselected devices, there is no way to exactly determine the value of R_D. However, if there exists a way to encode each memory line, where the parallel resistance sum of these unknown R_{U1-3} becomes known, this will yield the circuit as shown in Fig. 3C. In this case, the output voltage becomes deterministic, yielding only one logic '1' voltage, and one logic '0' voltage. The next section presents an encoding method to provide this solution.



(a) (b)

Fig. 4. A. The equivalent circuit when reading a '1' from a 4-bit line. B. The equivalent circuit when reading a '0' from a 4-bit line.

B. Balanced Encoding Crossbar Memory Data

In order to reliably read the selected device from the memory, a technique is needed that makes the output voltage deterministic, even though stray currents are present. Balanced encoding provides a solution to this situation. By definition, a balanced coded word is a binary stream that consists of an equal number of '1's and '0's. Encoding each word to have half ones and half zeros places half of the devices in each word into a high conductivity state, and half into a low conductivity state. Given the device resistances in each state and this encoding scheme, the lumped equivalent parallel resistance of the unselected devices can be easily calculated.

For example, given a four bit balanced coded word, assume the Device-Under-Test (DUT) (R_D in Fig. 3A) is a logic 1, which is represented by a capital R (logic 0 is represented by a lowercase r). This situation is described in Fig 4(a). Since the DUT is a logic 1, and the word is balanced coded, two of the unselected devices must be logic 0 and one unselected device must be a logic 1 yielding half 1s and half 0s. The order of the 1s and 0s does not matter, since now this resistance can be lumped together by the formula shown in Fig. 4(a). This situation also holds when reading a logic 0 represented by r, shown in Fig 4(b). Now there must exist two logic 1s and one logic 0 in the unselected devices for the encoding scheme to hold, and the equivalent resistance of these unselected devices is shown in Fig. 4(b). This will hold over all array sizes, where the formulas for the lumped unselected device resistance is as follows:

$$R_{equivOFF} = \frac{r}{\frac{N}{2} - 1 + \frac{r \cdot N}{2 \cdot R}} \tag{1}$$

$$R_{equivON} = \frac{R}{\frac{N}{2} - 1 + \frac{R \cdot N}{2 \cdot r}} \tag{2}$$

In these equations, N is the length of the encoded word. Eq. 1 shows the lumped unselected resistance when reading an off-state device, or a logic '0'. Eq. 2 shows the equivalent resistance when reading an on-state device. The critical takeaway point from this encoding solution is that the output becomes unique and deterministic. Since the current through the unselected devices is known, there exists only one logic '1' and one logic '0' output value, unlike before where the output was based heavily on the states of the unselected devices in the array. The output

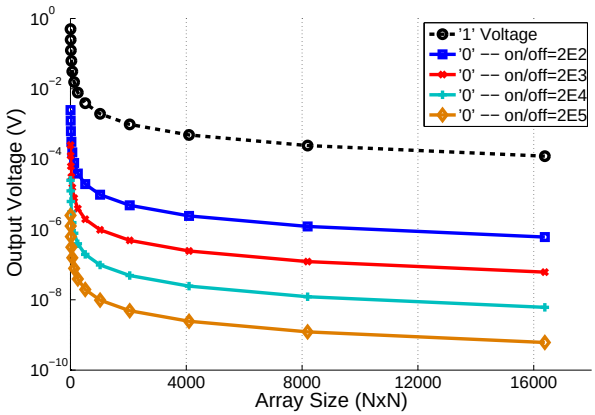


Fig. 5. Simulated result from Matlab showing the output voltage read from a logic '1' and a logic '0' while varying the device 'on/off' current ratio. Here the devices are assumed to just have fixed resistances based on their current logic state.

voltage can be calculated per the following equations.

$$V_{out1} = \frac{R_{equivON}}{R_{equivON} \cdot R_{on}} \cdot V_{RD} \tag{3}$$

$$V_{out0} = \frac{R_{equivOFF}}{R_{equivOFF} \cdot R_{off}} \cdot V_{RD} \tag{4}$$

The data encoding impact on array size is shown through the preliminary simulations in Fig. 5. This result is taken by simply implementing Eqs. 4 and 3 in Matlab, and observing the output voltages with different device 'on/off' current ratios. The dashed black line shows the logic '1' voltage read at the circuit output, while the remaining curves show how the logic '0' voltage changes with changing 'on/off' ratios.

As is observed, there exists only one output voltage per array size for each a logic '1' and '0', making the output voltage completely deterministic. Furthermore, this simulation shows how the output '1/0' voltage ratio stays well above 1 for all array sizes and all device 'on/off' current ratios. This implies that no matter the array size, it is always possible to distinguish a logic '1' from a logic '0' when reading any device from the crossbar array. With successful preliminary results, the next section presents more detailed circuit analysis to further validate the proposed encoding solution.

IV. Detailed Circuit Crossbar Simulations

A. Modeling Considerations for the Crossbar Array

Wire sizes projected for use in the crossbar architecture range anywhere from 5-50nm, with spacing between wires of similar magnitudes. Wires of this size are considered to be *nanowires*, and at these extreme interconnect densities, heavy emphasis is placed on accurately modeling their properties. This includes the nanowire resistance, the capacitance between each nanowire, and the capacitance from each nanowire to ground.

Nanowire Size	15nm
Nanowire Pitch	30nm
Substrate Cap. per Length	2 $\frac{pF}{cm}$
Coupling Cap. per Length	2 $\frac{pF}{cm}$
Nanowire Resistivity	8.88 $\mu\Omega$ cm

TABLE I
INTERCONNECT PARAMETERS USED FOR SIMULATING THE CROSSPOINT ARRAY BASED MEMORIES.

Nanowire crossbar capacitance modeling was recently discussed in several papers focused on using crossbar arrays for reconfigurable logic [16], [17], where the authors assume nanowire widths of approximately 15 nm, with similar spacings between the interconnect. Using some basic assumptions about wire geometry and dielectric materials, the authors arrive at a wire capacitance of 2 pF cm^{-1}. This same assumption is used here for both the substrate capacitance and the wire-to-wire coupling capacitance. While not all architectures have achieved yet such a high density (e.g. MRAM and phase change memory), the eventual goal is to shrink to such dimensions, therefore making this is an appropriate, albeit slightly pessimistic assumption in terms of impact on performance. Table I shows all of the interconnect model parameters that we use.

When modeling the resistance of such tiny nanowires, subtleties that have long been ignored start to have greater impact. Two such subtleties are surface and grain boundary scattering. One proposed method to model these nanoscale wires is presented in Steinhögl et. al. , where the effects of surface scattering and grain boundary scattering are combined according to Matthiessen's rule to achieve a unified resistance model [19]. Another way would be to empirically model actual resistance data taken from experiments as performed by Shi et. al [20]. Here, the resistance of such nanowires is modeled using the method presented by Steinhögl et. al. [19], achieving a resistivity of 8.88 $\mu\Omega$ cm for a 15 nm wide wire.

This section introduced several subtleties in modeling interconnects in nanowire crossbar arrays. The next section focuses on the actual devices interconnected between such dense nanowires. Given strong models for both the nanowires and nanodevices, it will be possible to create detailed circuit simulations of the crossbar array memories using SPICE based simulators.

B. Modeling Nano-Devices with the UDM

As previously mentioned, the memory types studied here span MRAM, PCM, and molecular memory. This work also considers some novel metal-oxide based devices, which have very recently been proposed for use in crossbar non-volatile memory [8], [9]. To study these different technologies, models for each device are required to simulate and evaluate the memories at a circuit level. Since these devices are typically non-linear and asymmetric, as was shown in Fig. 2B, a unique modeling tool is required to create SPICE level model files from this extracted data.

The Universal Device Model (UDM) provides a modeling platform to fit this need [22]. The UDM is capable of taking a set of data from a two-terminal device, and generate the appropriate fit equations and parameters needed

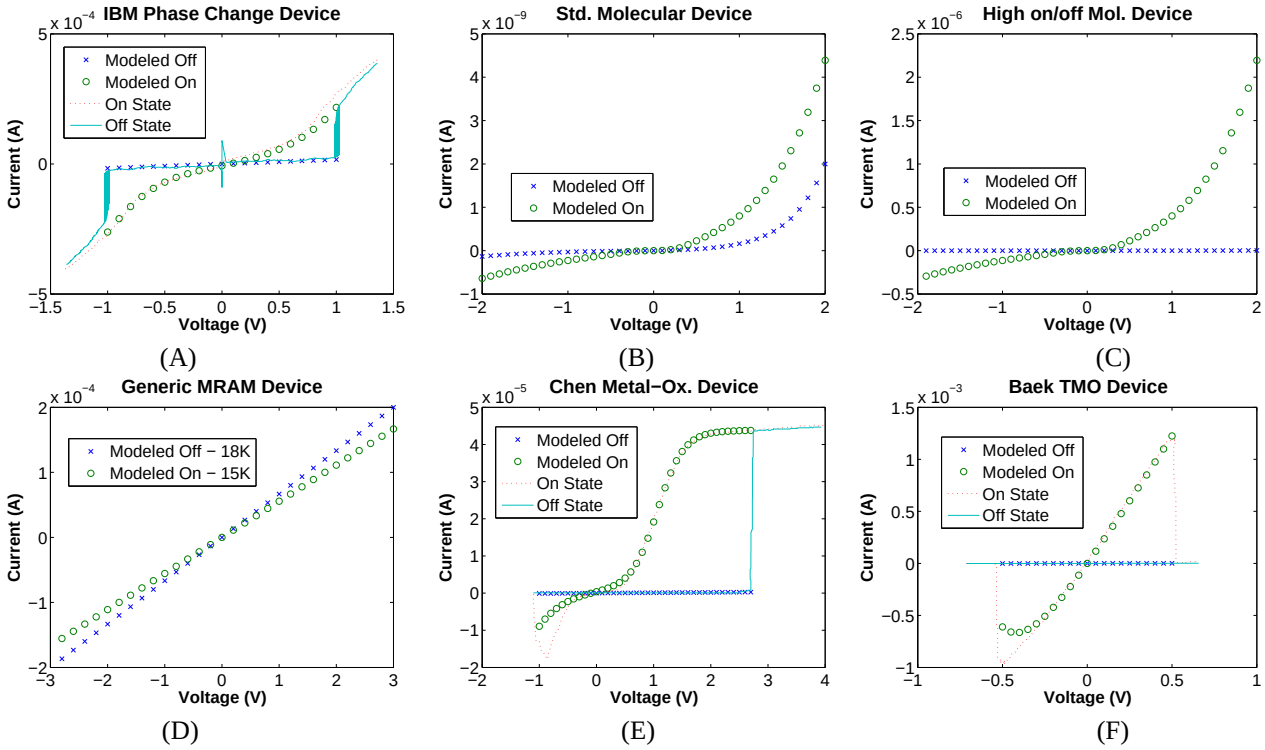


Fig. 6. (A) GeSb phase change device data device from IBM et. al. overlaid by the UDM modeled fit [21]. (B) Modeled data of a typical hysteretic molecular device. (C) Modeled data of a hysteretic molecular device with high on/off ratio ($\tilde{1}000$) (D) Modeled data of a generic MRAM device. (E) Data from a Cu_xO metal-oxide device overlaid by the UDM modeled fit [9]. (F) Data from a transition metal oxide (TMO) device overlaid by the UDM modeled fit [8].

to model the device. It does this by extracting certain nano-device characteristics from the empirical device data such as linear regions (resistor-like), thermionic emission (diode equation), resonant tunneling (Gaussian equation) and coulomb blockade (step function). With this analysis, it then uses a curve fitting routine to match the device data to a set of equations representing these typical properties, which are contained within the model itself. It then can generate VerilogA files useful for simulations with SPICE and Spectre [22].

Fig. 6 shows the devices modeled for use in this study. Six devices have been chosen: two molecular devices, two metal-oxide devices, one MRAM device, and one phase change device. The UDM modeled device data is shown, and in certain cases is overlaid by the actual device data extracted from original sources. If no actual device data is shown, it means that this device represents a generic IV characteristic for a broad set of these structures, such as the MRAM and molecular devices.

Fig. 6A shows a PCM device modeled from data taken from a recent work by IBM and partners [21]. The measured data is shown by solid lines while the modeled data is marked with circles (on-state) and x's (off-state). Similar data is shown in part B and C, except these are generic molecular devices. The characteristics modeled here generically represent the features found in many different works on molecular devices [12], [23], [24]. The device shown in Fig. 6c has a much higher on/off current ratio, more indicative of recent progress made with this technology [25].

Fig. 6D shows a modeled magnetic tunnel junction (MTJ) device for MRAM. The MTJ was modeled as a two-state linear resistive device. Studies have shown that MTJs are slightly non-linear over an applied voltage range, however these non-linearities are typically small, approximately 5 kΩ/V, and both states shift their impedances proportionally (i.e. if the on-impedance increases, the off-impedance correspondingly increases) [26]. The device impedances of 15 kΩ/ and 18 kΩ/ were selected from measured data [27]. The last two devices, Figs. 6E and 6F, modeled are metal-oxide based devices [8], [9].

C. Detailed Circuit Encoding Results

The encoding algorithm has shown to be effective in preliminary simulations, so to further examine this technique, detailed circuit-level simulations are performed on several encoded crossbar array structures. The crossbar array and nano-devices are modeled for circuit simulations based on the previously presented methods. The crossbar array is assumed to have a 30nm pitch, and six different devices are used to test the encoding effectiveness. Arrays are only simulated out to 64×64 as beyond this point, simulation times become extremely long. The specific simulator used is Cadence Virtuoso.

Fig. 7 shows several simulated results for both the encoded and unencoded memory designs. These simulations only consider the crossbar memory array itself, not the surrounding address and decoder logic. The peripheral logic used to read and write such nanoscale memories is presented in [6], and is the same for both the encoded and unencoded memory arrays. Therefore the overhead, in terms of power, performance, and area, will be similar no matter the employed design. The extra overhead needed to actually encode and decode the bits is presented in future sections.

Figs. 7A and B begin by showing how the encoder impacts the overall ’1’ to ’0’ output ratio as the memory array scales to large sizes. Fig. 7A shows both the encoded and unencoded ‘1/0’ ratios on the same plot, and as can be observed, the encoded ratios stay above 1 all the way out to at least a 64×64 memory array, while the unencoded ratios quickly fall below 1. Fig. 7B zooms in on the unencoded ratios, just to show where exactly these memory arrays fail. The unencoded memory also assumes that the unselected columns are left floating during the read operation.

Interestingly, the unique features of the different devices tend to make the ’1’/’0’ ratios scale at different rates for the encoded memories. For certain devices, it appears that the encoded memory could potentially scale out well beyond 64x64, such as the molecular devices, the phase change device, and the MRAM cell. The high on-off molecular devices achieve $F_{1/0}$ ratios as high as 1000. However the ‘1/0’ ratios for the metal-oxide devices scales down towards 3 or 4 as the array size approaches 64x64. The different scaling for different devices in not fully understood at this time, however could certainly be a starting point for future work in this area.

Furthermore, Figs 7C-E discuss several static power results for the proposed encoding solution. Static power is considered simply because the memory is essentially an array full of resistive elements, where applying a voltage across these devices will create quite a large static power draw, much like that of a resistive divider. As the circuit level solution requires grounding the unselected rows and columns, it is intuitive to think that this will increase the

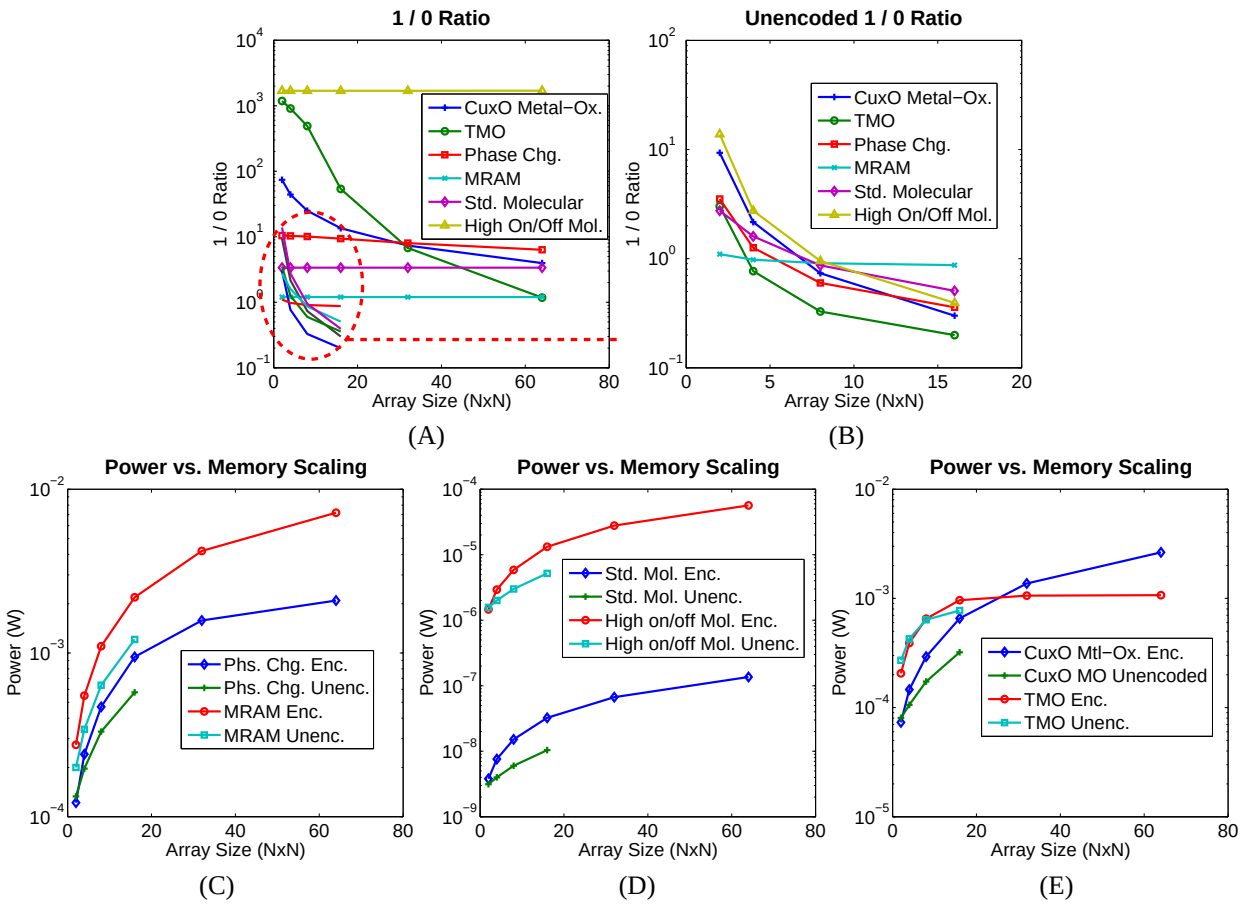


Fig. 7. A. Scaling of on/off ratio scales with increasing memory size. Each encoded memory can scale out to at least 64x64 while maintaining an on/off ratio of greater than 1. B. Details of the unencoded memory section in Fig. A. No unencoded memory can scale up past a 4x4 size. C.-E. Power tradeoff between the encoded and unencoded versions of each crosspoint memory. Approximately 1.5-3 times more power is required to read from the encoded memory compared to the unencoded memory. This does not take into account banking overhead from interconnect or extra decoder blocks.

overall static read power.

To examine this, this peak read power is plotted versus array size for each device under test. As is observed, the peak read power for each encoded memory is slightly higher than its unencoded counterpart. The data shows the power for the encoded memory to be anywhere from 1.5 to 3 times larger than the power for the unencoded memory, depending on the particular array size. Although this is an intrinsic drawback to this circuit/encoding technique, on nice thing about all of these non-volatile memory designs is that no voltage is required to hold the state of the memory elements. This means that this energy is consumed only when reading or writing the array elements, and then no static-energy is consumed when the memory is idle.

These sections have highlighted the benefits of the encoding solution, and shown how this technique works through simulation. However, there are many details needed to describe how one might use this technique within an actual nano-memory system. The next sections start by describing an algorithm to implement such a balanced encoding scheme in hardware, and will continue to discuss the overhead required for such an implementation.

N	M	Overhead
4	8	100%
8	14	75%
16	24	50%
32	42	31%
64	76	19%
128	142	11%

TABLE II
NUMBER OF BITS M REQUIRED TO BALANCE CODE A WORD OF LENGTH N.

V. HARDWARE IMPLEMENTATION OF THE ENCODER/DECODER

A. Balanced Encoding Algorithm

Now that the balanced encoding scheme has proven to be useful in circuit simulations, it is imperative to find an efficient method to perform the encoding and decoding in hardware. Balanced encoding was first introduced by Donald Knuth in 1986 [28], where he presented two fairly simple algorithms that produce balanced binary words from random data. Since then, many subsequent algorithms have been presented, generally aiming to improve the efficiency of the original algorithms, and reduce the number of overhead bits needed to code each word [29]–[31].

The algorithm chosen for use in this work is based on the parallel scheme described in Knuth's original work [28]. This method is primarily chosen for its simplicity and its conduciveness to hardware implementation, as will be seen in future sections. Other schemes yield balanced words, but require more computational overhead to encode and decode, which leads to more on-chip area devoted to the encoding design. Knuth's original parallel algorithm is as follows [28]:

1) Count the number of 1's in the word.
  - '10110111' - Has 6 '1's [110].

2) Serially flip each bit until the word contains half '1's and '0's. Count the number of bits that were flipped.
  - Resulting word - '01000111' : Count - '100'.

3) Attach count, followed by $\overline{count}$, to the new word.
  - Final word - '01000111[100][011]'.

By following this algorithm, it is possible to take any word and generate its balanced coded counterpart. For a word of length 2^n, the overhead involved with this scheme will always be an additional $2 \cdot n$. The overhead over various array sizes is shown in Table II. This table shows that as the word length grows, the percentage overhead involved with this scheme decreases. This makes it beneficial to use larger words for encoding and decoding to keep the area overhead to a minimum.

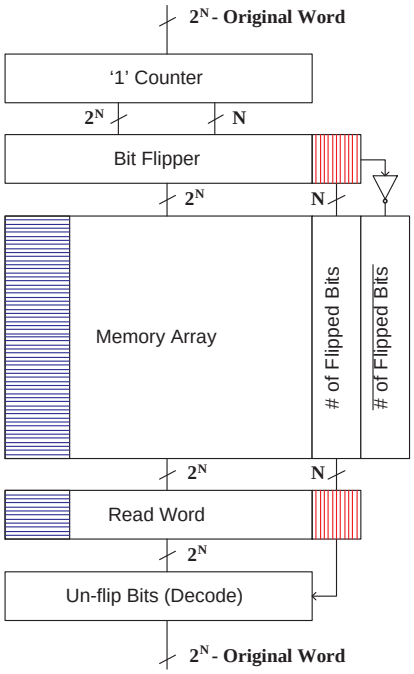


Fig. 8. A block diagram representing the steps as itemized in Sec. V-A. The filled blocks represent the number of bits flipped to achieve the coded word.

B. Architectural Implementation

The presented crossbar nano-memories have several appealing properties for architecture designers, including non-volatility, scalability, and low-power operation. For most modern computing platforms, it can be argued that memory is the primary performance bottleneck. Any reduction in memory latency can drastically increase the processor performance in terms of cycles per instruction (CPI). Furthermore, memory often occupies upwards of 50% of the processor die area, and can dominate the on-chip standby power consumption. It is imperative to minimize the overhead from these three sources : timing, performance, and power.

There exist both software and a hardware methods of performing this encoding/decoding. It is certainly feasible to write a software driver to perform the encoding/decoding functions. The primary overhead penalty associated with this method is performance latency, as the processor must use extra cycles to encode/decode, which it could be using for other functional tasks.

It is also feasible to create a dedicated piece of hardware in CMOS to perform this encoding and decoding. In general, creating this dedicated piece of hardware will elicit much faster processing, since this hardware can be custom tailored to achieve high performance for a given process. However, this comes with an area overhead penalty involved with the CMOS logic. For this work, the design is implemented in physical silicon, as it does minimize the latency penalty, which is vital to high performance memory designers.

A block diagram of the total encoding and decoding process is shown in Fig. 8. This diagram visually illustrates the encoding algorithm introduced in the previous section. Examining the diagram, the first step is to count the

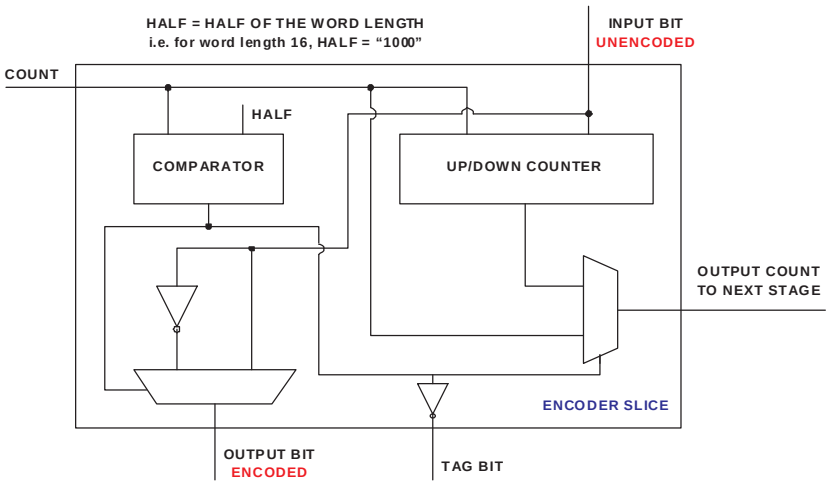


Fig. 9. A block level view of one slice in the balanced encoder datapath. This block accepts the current one’s count and data bit, and determines whether to flip this bit or leave it unchanged based on the one’s count.

number of ’1’s in the word. This count, along with the original word is fed into a ’bit flipper’, which implements procedures (2) and (3) in the algorithm described the previous section. Once encoded, this word is written into memory. On the decode side, the word is first read from the memory, and then the count attached to the word is used to determine the number of bits to flip back to obtain the original word (performed in the Decode block). In this figure, the vertical fill lines represent the area consumed by the binary count tag. The horizontal fill line represent the potential bits that were flipped.

While several methodologies exist to implement such an encoder/decoder design, this work employs a structural VHDL design strategy. This methodology yields a slightly less optimized design from a true custom ASIC circuit, however saves in design time by allowing synthesis tools to generate the final circuit layout. Furthermore, the VHDL can be compiled into FPGA architectures, which can allow quick calculations for expected performance, power, and area overheads.

Fig. 9 shows the detailed structural architecture of one encoder slice used in what is referred to as the ’Bit Flipper’ in Fig. 8. Each slice takes in the input count, which is the current number of ’1’s in the word, and the un-encoded input bit. The comparator compares the current input count to half of the length of the original input word. If this compare is false, this means the word is unbalanced, and does not have half ’1’s and half ’0’s. If this compare is true, then the word is balanced. Given the compare is false, the current input bit is flipped and fed to the output, and the up/down counter increases/decreases the ’1’ count based on which way the input bit flipped. This count is then fed to the next stage. If the compare returned true, the original input bit is fed to the output without being flipped, and the input count (which happens to equal half of the length of the input word) is fed directly to the output without being updated.

This slice is then included into a larger datapath, where there is one slice per bit in the memory line. This is shown in Fig. 10. This figure represents the architecture of the encoder. First, the number of ’1’s in the input is

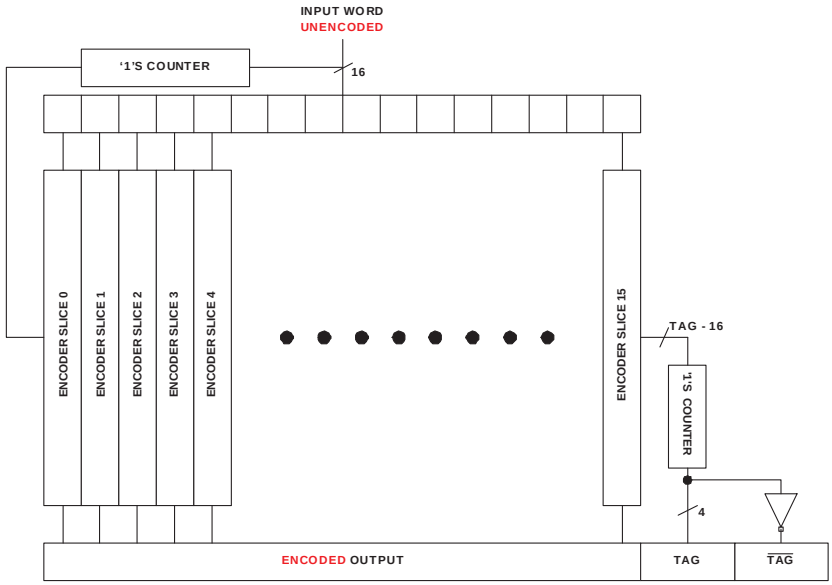


Fig. 10. A block view of the entire encoder datapath. Once performing the initial one's count, the word is serially processed to determine how many bits to flip to balanced code the word. The final count of flipped bits is attached to the end of the balanced word so it can be later decoded.

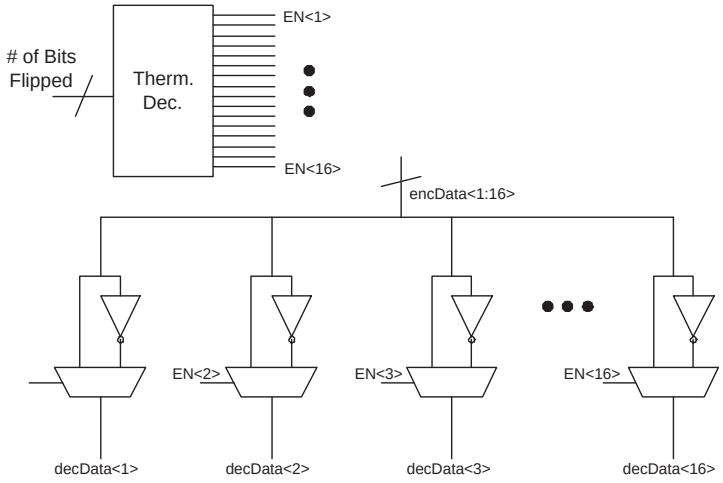


Fig. 11. Block diagram of the decoder design. The decoder simply flips back the bits that were initially flipped to create the balanced coded word.

determined by the '1' counter, and this count is fed into the first slice of the datapath. Then each slice determines whether or not the count has converged to half '1's and half '0's, and outputs the encoded word. These slices also output a tag, which is nothing more than the number of bits that were flipped to achieve the encoded output. This tag, along with its inverse, is attached to the end of the encoded word. The inverse of the tag is attached is to maintain the correct number of '1's and '0's, and it makes it easy to determine whether there is a bit error in the tag. Simply XORing each bit in the tag with each bit in the inverse tag should yield a '1' if there is no bit error. If the result is '0', this means there is a bit error in the tag.

Finally, Fig. 11 shows the structural implementation of the decoder logic. The decoder simply needs to invert the first N bits that were flipped in the encode process back to their original state. As the number of flipped bits N is actually stored in the word when encoded, it's easy to perform this procedure. The flipped bits N are fed into a thermometer decoder, which provides the enable signals to the decoding logic. Given the number of flipped bits as '101', the thermometer decoder outputs '11111000', indicating the first five bits need to be inverted back. The decode structure is straightforward, as multiplexors determine whether the output bits are flipped or not based on the enable signals from the thermometer decoder.

VI. Final Design Synthesis

As a final verification of the encoder/decoder design, each component is synthesized from VHDL to physical logic gates, using both FPGA and ASIC synthesis tools. The FPGA synthesis flow uses the free Xilinx ISE 10.1 tool, as can be obtained from the Xilinx website. The ASIC synthesis flow uses the Cadence toolset, including the specific tools RTL Compiler and Encounter, and yields the critical performance metrics of combinational path delay, total logic area, and total power consumption.

Table III shows some of the results from the design synthesis. The ASIC design is synthesized in a commercial 130nm technology, while FPGA results are given over several technology nodes. Comparing the two 130nm designs, the ASIC encoder is slightly more than 2X faster than the FPGA encoder, while the ASIC decoder is approximately 3X faster than the FPGA decoder.

Area results are only shown for the ASIC design, as the FPGA only reports the number of Combinational Logic Blocks (CLBs) used, and does not indicate the total CMOS area associated with such elements. The encoder design is approximately six times larger than the decoder design, as is expected from the more complex bit-sliced design structure. Furthermore, the encoder area is approximately equivalent to a $68 \times 68 \mu M^2$ square, where the decoder is equivalent to a $27.5 \times 27.5 \mu M^2$ block.

The synthesized ASIC encoder consumes a total average power of $626.3 \mu W$ including $1.5 \mu W$ of static power, yielding approximately $8.1 pJ$ per encode operation. Likewise, the ASIC decoder consumes a total of $31.44 \mu W$ of power, including $.24 \mu W$ of static power, yielding approximately $81.7 fJ$ per decode. Again, the power is not reported for the FPGA design, as it is difficult to determine which portion of the power comes from the utilized CLBs, and which comes from the static, unused CLB elements.

A. Discussions on Design Synthesis

The synthesis and simulation results show the encoder and decoder logic structures to work functionally and efficiently. However, one primary concern with such a design is that the encoder/decoder structures lie on the critical path to and from the memory. Particularly for the encoder, adding an additional 12.9ns could pose a huge timing overhead into the overall processor system delay. Although it generally takes several clock cycles to access data from main memory, assuming a 2GHz clock, 12.9ns equates to 26 clock cycles, which is a long time to wait for data to return.

		Xilinx FPGA		
	ASIC	Virtex-2	Virtex-4	Virtex-5
Tech. Node	130nm	130nm	90nm	65nm
Encoder Delay (ns)	12.9	29.6	24.8	17.25
Encoder Area (μM^2)	4569	-	-	-
Decoder Delay (ns)	2.6	8.3	6.7	5.23
Decoder Area (μM^2)	756	-	-	-

TABLE III
SYNTHESIS RESULTS COMPARED BETWEEN AN ASIC AND AN FPGA IMPLEMENTATION OF THE 16-BIT ENCODER AND DECODER DESIGNS. THE ASIC RESULTS GIVEN ARE FROM THE RTL SYNTHESIS LEVEL, AND THE FPGA RESULTS ARE BASED UPON THE XILINX TOOL ESTIMATES AFTER PLACE-AND-ROUTE.

One potential solution to this long delay is to pipeline the encoder. One nice property of this particular topology is that it would easily lend itself to a pipelined design, where each slice would be clocked, and any memory write data access could stream through the encoder. Much like a traditional pipelined design, this would heavily increase the overall throughput of the encoder. While the first write would take 26 cycles, the next data would come out of the encoder in $\frac{26}{16}$ cycles. After rounding up, 2 extra cycles are needed for the next data to appear ready to write. Of course, some additional design is needed for this proposed pipelined encoder, and is not presented in this work. This is one area of interest for continued future work on this topic that could provide a strong solution for writing to resistive memories.

This similar pipelining concept could not be applied to the decoder design, however the decoder only takes the equivalent of 6 $2GHz$ clock cycles to complete its task. Furthermore, as the decoders are much smaller, several of these could be placed in parallel to increase the overall throughput of the design.

VII. COMPARISON STUDY AGAINST MRAM

A. 1T-1MTJ Read and Write Overhead

Previous sections have introduced a method to encode and decode memory words in resistive memories. With this encoding scheme, the pieces are in place to generate a full nano-scale crossbar memory, which can scale to large sizes despite the negative impacts of stray currents. However, to justify the usefulness of this scheme, is it pertinent to compare the results of such a system against state-of-the-art non-volatile memory designs. This section compares the proposed resistive memory scheme against the popular 1-Transistor 1-Magnetic-Tunnel-Junction (1T-1MTJ) MRAM design.

The 1T1MTJ design is an alternate design to the crossbar memory introduced throughout this work. In the 1T-1MTJ topology, Each bit-cell is fixed with one transistor so that one can uniquely read and write each bit. A more detailed view of the read and write circuitry is shown in Fig. 12, which is adapted from the work by Maffitt et. al. in [32]. The selected bit is highlighted in the bottom right portion of the figure.

In order to determine the state of a particular bit in the array, it is necessary to determine the resistance of the selected bit. The word-line (WL) is used to select the proper row in the memory array, and the column decoder

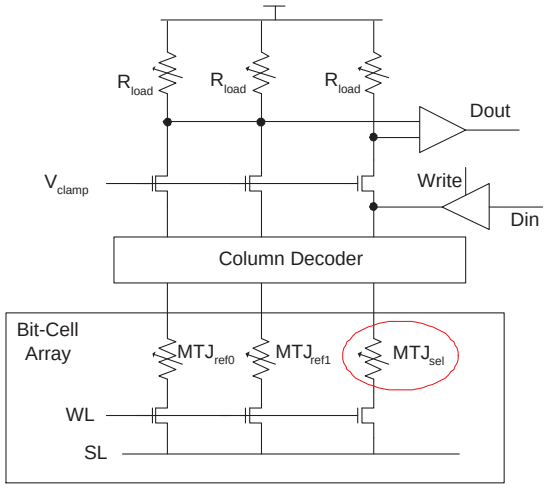


Fig. 12. The peripheral circuitry used to both read and write a 1T-1MTJ bit cell. Adapted from [32].

ensures the selected bit is being read, not other unselected bits on the row. The resistance of any bit, whether in the high or low conductivity state, can vary based on property variations within the tunnel-junctions. Therefore, to determine the resistance state, a voltage sense amplifier is used to compare the selected bits resistance state against a reference voltage.

The reference is created by averaging the resistance of two opposing state MTJ devices (MTJ_{ref0}, MTJ_{ref1}), which creates a voltage lying halfway between that of an observed '1' or '0' value. These reference devices can be shared among arrays, and for the comparison here, it is assumed that there exists a unique MTJ_{ref0} and MTJ_{ref1} for each bit read from the bank (i.e. a 16 bit word needs 16 reference MTJ pairs). The NMOS devices gated by V_{clamp} act as source followers, which ensure that the reference and selected bits are biased at the same voltage level. The load devices then act as a resistive divider to help to transform the current through the MTJs into a voltage for the sense amplifier. The average reference voltage is generated by tying the NMOS drains of both the '0' and '1' reference MTJs together [32]. A more detailed description of how the circuitry works can be found in the original work [32].

Writing the selected MTJ is performed simply by properly addressing the selected bit through the row/column decoder, and then placing the proper values on the Din input and the Source-Line (SL) wire. Bi-directional current flow is often required to program such devices into parallel/anti-parallel states. To accomplish this, the SL driver must be configurable to drive either a high voltage or ground.

B. Quantitative Area Comparison Against MRAM

The read/write circuitry for crossbar based nanoscale memories is discussed in previous works [3], [6]. Per row, the logic requires two multiplexors, one inverter, one PMOS device, and $log_2(N) + 1$ NMOS devices, where N is the number of address bits. Three multiplexors and two inverters are required per column. A standard transmission gate based column decoder is implemented, where the number of gates scales logarithmically with the size of the

array. An extra two NOR gates and two multiplexors are required to control the load devices and multiple clocks. This overhead can further be explored in the original works from Rose et. al. [6].

Given the basic MRAM design introduced in the previous section, it is now possible to give some quantitative overhead analysis of the two separate nano-memory implementations. For both memories, the row and column decoders are implemented in a similar fashion. The row decoder is a typical one-stage NMOS decoder. The column decoder is a transmission gate based multiplexor design, as the signals traversing through this decoder are not digital in nature. The area overhead comparison is made here considering both a 65nm and a 130nm CMOS technology for the peripheral circuit implementations. The MRAM bit cell is assumed to be $0.3584\mu m^2$, as it is the smallest demonstrated experimental MRAM cell the authors found described in the literature [33]. The resistive memory device is a molecular memory device selected from [2], and is sized $.0011\mu m^2$. This again is the smallest experimentally demonstrated resistive memory cell found in the literature.

Fig. 13A shows the area overhead comparison between the MRAM design and the molecular memory designs given a 130nm CMOS technology. While it is apparent that the molecular memory design is smaller than the MRAM design, adding the encoder/decoder design introduces a large overhead. The encoder/decoder is a 16-bit design, and it is assumed that only one encoder/decoder is used per bank, which is reasonable since only one word can be read/written at a time. Given a 130nm peripheral circuit implementation, the molecular memory with this encoder/decoder doesn't actually achieve any area savings over the MRAM until the bank size is at least $16kb$ large, or $128x128$ bits. This is the breakeven point for the design. Beyond $16kb$, it is apparent that even with the encoder/decoder, the molecular memory is still much smaller than the MRAM design.

Fig. 13B shows the same design comparison except given a 65nm implementation of the CMOS peripheral circuits. In this figure, the breakeven point actually shifts back $4kb$ (64 x 64 bits). This is largely because the memory array sizes are fixed, and the dominant encoder/decoder area is scaled back by typical CMOS scaling rules ($1/\sqrt{area}$ per technology node). Here the molecular memory, even with the encoder/decoder design, is approximately $41X$ smaller than the MRAM design assuming a $512x512$ bit memory bank.

Figs. 14A,B delve further into the overhead to highlight the primary area sources within each design. Fig. 14A shows the area breakdown for the MRAM design, and it is apparent that area is dominated by the bit-cell array. This makes sense as the bit cell is over 100 times larger than the molecular memory bit cell. Fig. 14B shows the opposite result for the molecular memory, where while the bit-cell array does contribute significantly to the total area, the majority is consumed by the peripheral circuits. This is particularly true at larger array sizes, however at these large array sizes, the overall encoder/decoder weighs less into the total overhead.

C. Final Comparison Thoughts

The previous section highlighted the area overheads with both the MRAM and molecular memory technologies. While its hard to properly quantify the overheads in terms of power and performance without a full MRAM design, it is possible to at least qualitatively discuss this topic.

Previous sections highlighted the energy per encode/decode given a 16-bit design, and the numbers came out to

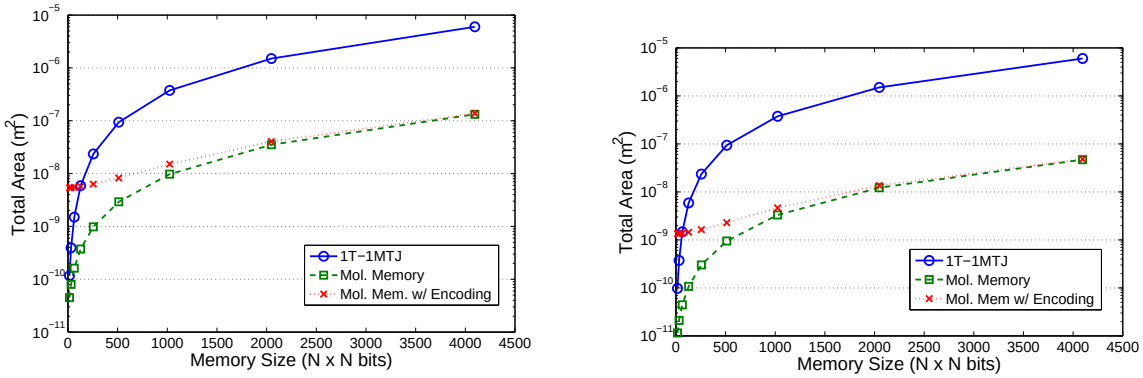


Fig. 13. A. Area overhead comparison between 1T-1MTJ MRAM and crossbar molecular memory, using 130nm CMOS for peripheral circuits. B. Same overhead comparison between MRAM and molecular memory, except using 65nm CMOS for peripheral circuits.

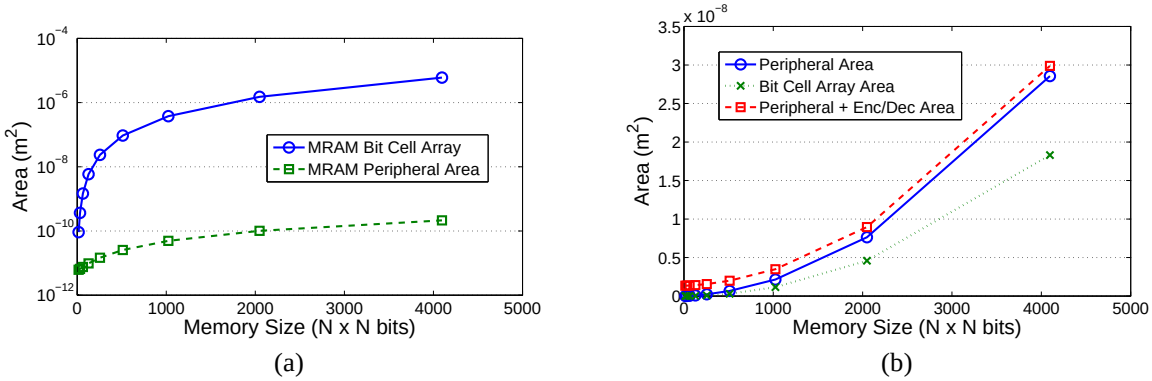


Fig. 14. A. Area breakdown for the 1T-1MTJ MRAM design. The bit-cell array dominates the total memory area. B. Area breakdown of the molecular memory. The peripheral logic heavily factors into the total area, particularly when considering the physical encoders/decoders.

approximately $8.1pJ$ per encode, and $81.7fJ$ per decode. The 2009 ITRS states that the best currently demonstrated MRAM write energy per bit is $150pJ$ [34]. Given a 16-bit word, this would yield $2.4nJ$ per write given the best currently demonstrated MRAM. Ignoring the very low decode energy for the current design, to encode a 16-bit word would consume approximately $130pJ$, approximately 18.5 times less than when writing to an MRAM (read energies not given in ITRS). Furthermore, the projected write energy per bit for such a molecular memory is $2 * 10^{-19}J$. This value is obviously a projected value, and not something experimentally measured, however by observing the numbers, it is fairly clear to see that the molecular memory implementation will likely yield a significant power savings over the MRAM design. This is even considering the encoder/decoder.

Although the molecular memory can be configured to consume less power and use less area, it cannot match the performance of modern MRAM circuits. Earlier sections highlighted the performance penalty incurred by both the encoder and decoder, $12.9ns$ and $2.6ns$ respectively. Furthermore, the delay in reading and writing the molecular devices themselves is quite long, hovering around $10ns$ and $40ns$ respectively, according to the newest ITRS [34]. On the other side, MRAM can typically be read and written to in less than $.5ns$. The takeaway point from this

section is not that the molecular memory is slow, however that there is an important tradeoff between performance, power, and area to consider before choosing any one of these novel nano-electronic memory solutions.

VIII. Conclusion

This work presents a circuit and data encoding method to effectively reduce the impact of stray current paths in crossbar array memories. The presented solution allows crossbar memories to scale out beyond the 64x64 array size. The balanced encoding method introduced for nanoscale memories ensures the voltage output from any selected device is deterministic. Comparison results are presented against a common 1T-1MTJ architecture, showing a strong area and power reduction over MRAM, however suffering in performance capability.

Future work will focus on improving the hardware implementation, particularly on pipelining the physical encoder design. This design change will elicit a higher overall throughput, and decrease the performance penalty seen through the encoder. Further work will examine the impacts of device variations on the encoding technique.

Acknowledgment

The authors would like to thank J. Huang, W. Huang and Z. Qi from the University of Virginia for both help with synthesizing the logic circuits, and interesting discussions on this topic.

References

[1] "International technology roadmap for semiconductors 2007 edition."

[2] J. E. Green, J. W. Choi, A. Boukai, Y. Bunimovich, E. Johnston-Halperin, E. Delonno, Y. Luo, B. A. Sheriff, K. Xu, Y. S. Shin, H.-R. Tseng, J. F. Stoddart, and J. R. Heath, "A 160-kilobit molecular electronic memory patterned at 10^{11} bits per square centimetre," *Nature*, vol. 445, pp. 414–417, January 2007.

[3] G. S. Rose and A. C. C. et al., "Design approaches for hybrid cmos/molecular memory based on experimental device data," in *Proceedings Great Lakes Symposium on VLSI*, April 2006, pp. 2–7.

[4] M. M. Ziegler and M. R. Stan, "Design and analysis of crossbar circuits for molecular nanoelectronics," in *Proceedings IEEE Conference on Nanotechnology*, August 2002, pp. 323–327.

[5] ——, "CMOS/nano co-design for crossbar-based molecular electronic systems," *IEEE Trans. Nanotechnol.*, vol. 2, no. 4, pp. 217–230, December 2003.

[6] G. S. Rose, Y. Yao, J. M. Tour, A. C. Cabe, N. Gergel-Hackett, N. Majumdar, J. C. Bean, L. R. Harriott, and M. R. Stan, "Designing cmos/molecular memories while considering device parameter variations," *Journal of Emerging Technologies in Computing Systems*, 2007, accepted.

[7] K. Galatsis, K. Wang, Y. Botros, Y. Yang, Y.-H. Xie, J. F. Stoddart, R. B. Kaner, C. Ozkan, J. Liu, M. Ozkan, C. Zhou, and K. W. Kim, "Emerging memory devices: Nontraditional possibilities based on nanomaterials and nanostructures," *IEEE Circ. and Dev. Mag.*, vol. 22, no. 3, pp. 12–21, 2006.

[8] I. G. Baek, M. S. Lee, S. Seo, and M. J. L. et al, "Highly scalable non-volatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *Proc. of IEEE Elec. Dev. Mtg.*, December 2004, pp. 587–590.

[9] A. Chen, S. Haddad, Y.-C. Wu, and T.-N. F. et al, "Non-volatile resistive switching for advanced memory applications," in *Proc. of IEEE Elec. Dev. Mtg.*, December 2005, pp. 746–749.

[10] J. Maimon, E. Spall, R. Quinn, and S. Schnur, "Chalcogenide-based non-volatile memory technology," in *Proc. of IEEE Aero. Conf.*, 2001, pp. 2289–2294.

[11] F. Pellizzer, A. Benvenuti, B. Gleixner, Y. Kim, B. Johnson, M. Magistretti, T. Marangon, A. Pirovano, R. Bez, and G. Atwood, "A 90nm phase change memory technology for stand-alone non-volatile memory applications," in *Proc. of IEEE Symp. on VLSI*, 2006, pp. 122–123.

[12] N. Gergel, N. Majumdar, and K. K. et al, "Study of the room temperature molecular memory observed from a nanowell device," *J. Vac. Sci. Tech*, vol. 23, no. 4, pp. 880–885, August 2005.

[13] M. Durlam, B. Craigo, and M. D. et al, "Toggle mram: A highly reliable non-volatile memory," in *IEEE Int. Symp. on VLSI Tech. Sys. and Apps*, April 2007.

[14] J. Debrosse, D. Gogl, and A. B. et al, "A high-speed 128-kb mram core for future universal memory applications," vol. 39, no. 4, pp. 678–683, April 2004.

[15] K. K. Likharev and D. B. Strukov, "CMOL: Devices, circuits, and architectures," *Lect. Notes Phys.*, vol. 680, pp. 447–478, 2005.

[16] D. B. Strukov and K. K. Likharev, "CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, pp. 888–900, 2005.

[17] G. S. Snider and R. S. Williams, "Nano/CMOS architectures using a field-programmable nanowire interconnect," *Nanotechnology*, vol. 18, no. 035204, 2007.

[18] M. R. Stan, G. S. Rose, and M. M. Ziegler, "Hybrid CMOS/molecular electronic circuits," in *Proc. of Int. Conf. on VLSI Design*, January 2006, pp. 703–708.

[19] W. Steinhögl, G. Schindler, G. Steinlesberger, and M. Engelhardt, "Size-dependent resistivity of metallic wires in the mesoscopic range," *Phys. Rev. B*, vol. 66, no. 7, 2002.

[20] S. X. Shi and D. Z. Pan, "Wire sizing with scattering effect for nanoscale interconnection," in *Proc. of Asia and Sth. Pac. Des. Aut. Conf.*, 2006, pp. 503–508.

[21] Y. C. Chen, C. T. Rettner, and S. R. et al, "Ultra-thin phase-change bridge memory device using gesb," in *Proc. of IEEE Elec. Dev. Mtg.*, December 2006, pp. 1–4.

[22] G. S. Rose, M. M. Ziegler, and M. R. Stan, "Large-signal two-terminal device model for nanoelectronic circuit analysis," *IEEE Trans. on VLSI*, vol. 12, no. 11, pp. 1201–1208, November 2004.

[23] Y. Chen, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, D. L. Olynick, and E. Anderson, "Nanoscale molecular-switch devices fabricated by imprint lithography," vol. 82, no. 10, pp. 1610–1612, March 2003.

[24] C. P. Collier, G. Mattersteig, E. W. Wong, Y. Luo, K. Beverly, J. Sampaio, F. M. Raymo, J. F. Stoddart, and J. R. Heath, "A [2]catenane-based solid state electronically reconfigurable switch," *Science*, vol. 289, pp. 1172–1175, 2000.

[25] D. R. Stewart, D. A. A. Ohlberg, P. A. Beck, Y. Chen, R. S. Williams, J. O. Jeppesen, K. A. Nielsen, and J. F. Stoddart, "Molecule-independent electrical switching in pt/organic monolayer/ti devices," *Nano Letters*, vol. 4, no. 1, pp. 133–136, 2004.

[26] C.-H. Cho, J. H. Ko, and D. Kim, "A cmos macro-model for mtj resistor of mram cell," *Physica Status Solidi (a)*, vol. 8, pp. 1653–1657, June 2004.

[27] B. N. Engel, J. Akerman, and B. B. et al, "A 4-mb toggle mram based on a novel bit and switching method," *IEEE Trans. on Magnetics*, vol. 41, no. 1, pp. 132–136, January 2005.

[28] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. on Inf. Theory*, vol. IT-32, no. 1, pp. 51–53, January 1986.

[29] W. P. Cornelius and W. C. Athas, "Methods and apparatus for constant-weight encoding and decoding," Patent, December, 2003, u.S. Patent no. 6661355. Apple Computer, Inc.

[30] L. G. Tallini and B. Bose, "Balanced codes with parallel encoding and decoding," *IEEE Trans. Computers*, vol. 48, no. 8, pp. 794–814, August 1999.

[31] R. Mascella, L. G. Tallini, S. Al-Bassam, and B. Bose, "On balanced codes over the m-th roots of unity, m=4*," in *Proceedings IEEE International Symposium on Information Theory*, July 2003, p. 133.

[32] T. M. Maffitt, J. K. Debrosse, J. A. Gabric, E. T. Gow, M. C. Lamorey, J. S. Parenteau, D. R. Wilmott, M. A. Wood, and W. J. Gallagher, "Design considerations for MRAM," *IBM Journal of Research and Development*, vol. 50, no. 1, pp. 25–39, January 2006.

[33] K. Tsuchida, T. Inaba, K. Fujita, Y. Ueda, T. Shimizu, Y. Asao, T. Kajiyama, M. Iwayama, K. Sugiura, S. Ikegawa, T. Kishi, T. Kai, M. Amano, N. Shimimura, H. Yoda, and Y. Watanabe, "A 64mb MRAM with clamped-reference and adequate-referencing schemes," in *Proc. of Int. Solid State Circuits Conf.*, February, pp. 258–259.

[34] "International technology roadmap for semiconductors 2009 edition."